**3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

Рассмотрим подробнее функционирование программного модуля. В предыдущем разделе были описаны все блоки программного модуля. В этом разделе будут описаны классы, с помощью которых будет реализован функционал каждого модуля.

Диаграмма классов программного модуля для построения трёхмерных моделей объектов по изображениям приведена на чертеже ГУИР.400201.040 РР.1.

Далее рассмотрим каждый блок программного модуля и классы, которые в нём реализованы.

**3.1 Функциональное проектирование блока декодирования изображений**

Данный блок предназначен для преобразования изображений из формата, в котором оно было сохранено, в формат, пригодный для применения алгоритмов обработки.

Приложение будет поддерживать достаточно большое количество форматов изображений: bmp, dib, jpeg, jpg, jpe, jp2, png, webp, pbm, pgm, ppm pxm, pnm, sr, ras, tiff, tif, exr, hdr, pic. Этот список форматов во многом обусловлен форматами, которые поддерживаются в библиотеке OpenCV в модуле imgcodecs.

Особое внимание стоит обратить на форматы JPEG(jpeg, jpg, jpe) и JPEG2000(jp2). Так как декодирование изображений в OpenCV выполняется на CPU, оно не обладает большой скоростью. Для форматов JPEG и JPEG2000 компания Nvidia выпустила библиотеки, которые выполняют декодирование на GPU. Данные библиотеки называются NvJPEG и NvJPEG2000.

Сравним скорость декодирования изображений с помощью OpenCV и NvJPEG. Для этого выполним декодирование нескольких изображений и измерим время декодирования одного изображения (см. таблицу 3.1).

Таблица 3.1 – Время декодирования изображений OpenCV и NvJPEG

|  |  |  |
| --- | --- | --- |
| Номер изображения | Время декодирования OpenCV, мкс. | Время декодирования NvJPEG, мкс. |
| 1 | 8299 | 2820 |
| 2 | 8431 | 2811 |
| 3 | 8517 | 2830 |
| 4 | 9921 | 2717 |
| 5 | 8625 | 2802 |
| 6 | 8669 | 2605 |
| 7 | 8852 | 2543 |
| 8 | 8773 | 2452 |

Данный эксперимент проводился на CPU Intel Core i9-9900 и GPU Nvidia RTX 3080. Среднее время декодирования изображения средствами OpenCV – 8761 мкс, средствами NvJPEG – 2698 мкс. Как можно заметить, скорость декодирования с помощью NvJPEG выше более чем в 3 раза.

Для NvJPEG2000 можно добиться примерно того же результата.

Исходя из полученных результатов, следует отдельно выделить классы для декодирования изображений, которые используют GPU. При этом в силу небольшого числа форматов, которые можно декодировать на графическом процессоре, остальные форматы можно декодировать одним классом, основанным на OpenCV.

В таблице 3.2 приведены классы данного блока и их назначение.

Таблица 3.2 – Классы блока декодирования изображений и их назначение

|  |  |
| --- | --- |
| Название клсса | Назначение |
| IImageDecoder | Базовый класс для декодеров изображений. |
| ImageDecoderFactory | Класс, реализующий шаблон проектирования «фабрика» для создания различных декодеров. |
| NvJPEG2kImageDecoder | Декодер изображений в формате JPEG2000. |
| NvJPEGHardwareImageDecoder | Декодер изображений в формате JPEG, использующий аппаратное декодирование изображений. Доступно только для видеокарты Nvidia A100. |
| NvJPEGImageDecoder | Декодер изображений в формате JPEG. |
| OpenCVImageDecoder | Универсальный декодер. |

В таблицах 3.3 – 3.7 приведены поля и методы классов, реализованных в данном блоке.

Таблица 3.3 – Поля и методы класса IImageDecoder

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| IImageDecoder() | Конструктор по умолчанию. |
| virtual ~IImageDecoder() noexcept(false) | Виртуальный деструктор по умолчанию. |
| virtual bool Decode(const unsigned char\* data, unsigned long long size,CUDAImage& decodedImage) | Виртуальный метод, предназначенный для декодирования изображения. |
| virtual void Initialize() | Виртуальный метод, выполняющий инициализацию декодера. |
| virtual bool IsInitialized() | Виртуальный метод, проверяющий, инициализирован ли декодер. |

Данный класс является базовым для всех декодеров.

Таблица 3.4 – Поля и методы класса ImageDecoderFactory

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| static std::unique\_ptr<IImageDecoder> Create(DecoderType type, bool useCUDAStream, void\* cudaStream) | Данный метод возвращает указатель на созданный объект декодера, тип которого передаётся как аргумент данной функции. |

Таблица 3.5 – Поля и методы класса NvJPEGImageDecoder

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| 1 | 2 |
| explicit NvJPEGImageDecoder(cudaStream\_t cudaStream) | Конструктор. |
| ~NvJPEGImageDecoder() noexcept(false) override | Деструктор. |
| bool Decode(const unsigned char\* data, unsigned long long size, DataStructures::CUDAImage& decodedImage) override | Метод, предназначенный для декодирования изображения. Основан на методе DecodeInternal. |
| void Initialize() override | Метод, выполняющий инициализацию декодера. |
| bool IsInitialized() override | Метод, проверяющий, инициализирован ли декодер. |
| bool DecodeInternal(const unsigned char\* data, unsigned long long size, DataStructures::CUDAImage& image) | Метод, выполняющий декодирование изображение в формате JPEG на GPU. |
| virtual void AllocateBuffer(int width, int height, int channels) | Метод, выделяющий буфер для изображение в памяти GPU. |
| nvjpegJpegState\_t state\_ | Хранит промежуточную информацию этапов декодирования. |
| nvjpegJpegState\_t decoupledState\_ | Внутреннее состояние декодера. |

*Продолжение таблицы 3.5*

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| 1 | 2 |
| nvjpegHandle\_t handle\_ | Используется для внутренних нужд NvJPEG. |
| size\_t bufferSize\_ | Размер выделенного буфера. |
| bool initialized\_ | Флаг инициализации декодера. |
| bool InitDecoder() | Метод, инициализирующий декодер. |
| nvjpegBufferPinned\_t pinnedBuffer\_ | Используется для внутренних нужд NvJPEG. |
| nvjpegBufferDevice\_t deviceBuffer\_ | Используется для внутренних нужд NvJPEG. |
| nvjpegDecodeParams\_t decodeParams\_ | Используется для внутренних нужд NvJPEG. |
| nvjpegJpegDecoder\_t decoder\_ | NvJPEG декодер. |
| nvjpegJpegStream\_t jpegStream\_ | Хранит информацию о декодируемом изображении. |
| cudaStream\_t cudaStream\_ | Используется в асинхронных операциях копирования. |
| nvjpegImage\_t imageBuffer\_ | Буфер для изображения в памяти GPU. |

Таблица 3.6 – Поля и методы класса OpenCVImageDecoder

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| OpenCVImageDecoder() | Конструктор. |
| ~OpenCVImageDecoder() noexcept(false) override | Деструктор. |
| bool Decode(const unsigned char\* data, unsigned long long size, DataStructures::CUDAImage& decodedImage) override | Виртуальный метод, предназначенный для декодирования изображения. |
| void Initialize() override | Метод, выполняющий инициализацию декодера. |
| bool IsInitialized() override | Метод, проверяющий, инициализирован ли декодер. |

Таблица 3.7 – Поля и методы класса NvJPEG2kImageDecoder

|  |  |  |
| --- | --- | --- |
| Сигнатура поля/метода | | Назначение |
| explicit NvJPEG2kImageDecoder(cudaStream\_t cudaStream) | | Конструктор. |
| std::vector<unsigned char\*> bufferChannels\_ | Хранит указатели на участки памяти GPU для буферизации изображений |
| std::vector<size\_t> bufferChannelsPitches\_ | Флаг инициализации декодера. |
| std::vector<size\_t> bufferChannelsSizes\_ | Метод, инициализирующий декодер. |
| bool initialized\_ | Флаг инициализации декодера. |
| ~NvJPEG2kImageDecoder() noexcept(false) override | Деструктор. |
| bool Decode(const unsigned char\* data, unsigned long long size, DataStructures::CUDAImage& decodedImage) override | Виртуальный метод, предназначенный для декодирования изображения. |
| bool DecodeInternal(const unsigned char\* data, unsigned long long size, DataStructures::CUDAImage& outputImage) | Метод, выполняющий декодирование изображение в формате JPEG2000 на GPU. |
| void AllocateBuffer(int width, int height, int channels, size\_t elementSize) | Метод, выделяющий буфер для изображение в памяти GPU. |
| bool InitDecoder() | Инициализирует NvJPEG2000 декодер. |
| cudaStream\_t cudaStream\_ | Используется для асинхронных операций на GPU. |
| nvjpeg2kHandle\_t handle\_ | Используется для внутренних нужд NvJPEG2000. |
| nvjpeg2kDecodeState\_t decodeState\_ | Хранит промежуточную информацию этапов декодирования. |
| nvjpeg2kStream\_t | Хранит информацию о декодируемом изображении. |

**3.2 Функциональное проектирование блока управления графическими процессорами**

В этом блоке будут реализованы классы, которые содержат функционал выбора активного графического процессора.

Классы данного блока перечислены в таблице 3.8.

Таблица 3.8 – Классы блока декодирования изображений и их назначение

|  |  |
| --- | --- |
| Название клсса | Назначение |
| GPU | Класс, хранящий информацию о графическом процессоре. |
| GpuManager | Класс, отвечающий за выбор активного графического процессора. |

Поля и методы вышеперечисленных классов расписаны в таблицах 3.9 и 3.10

Таблица 3.9 – Поля и методы класса GPU

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| int deviceId\_ | Индекс GPU. |
| std::string name\_ | Название GPU. |
| int computeCapabilityMajor\_ | Старшая цифра версии Compute Capability. |
| int computeCapabilityMinor\_ | Младшая цифра версии Compute Capability. |
| int multiprocessorsAmount\_ | Количество потоковых мультипроцессоров на GPU. |
| std::size\_t memoryTotal\_ | Объём глобальной памяти на GPU. |
| double memoryBandwidth\_ | Пропускная способность памяти GPU. |
| int maxThreadsPerMultiprocessor\_ | Максимальное количество потоков, одновременно выполняемых на одном мультипроцессоре. |
| int maxThreadsPerBlock\_ | Максимальное количество потоков в CUDA блоке. |
| std::size\_t sharedMemPerBlock\_ | Максимальный объём разделяемой памяти в блоке. |

Данный класс описывает все CUDA-совместимые устройства системы.

Далее будет описан класс GpuManager – класс, отвечающий за выбор активного GPU для исполнения алгоритмов фотограмметрии.

Таблица 3.10 – Поля и методы класса GpuManager

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| void UpdateCUDACapableDevicesList() | Обновляет список графических процессоров и записывает данные в cudaCapableDevices\_. |
| unsigned int GetCUDACapableDevicesAmount() | Определяет число графических процессоров. |
| const std::vector<GPU>& GetCUDACapableDevicesList() | Предоставляет доступ к элементу cudaCapableDevices\_. |
| GPU& SelectMatchingGPU(const std::shared\_ptr<Config::JsonConfig>& serviceConfig) | Выбирает GPU с учётом заданной политики. |
| void SetDevice(GPU& gpu) | Назначает выбранный GPU активным. |
| const std::shared\_ptr<GPU>& GetCurrentGPU() | Возвращает указатель на текущий активный GPU. |
| std::vector<GPU> cudaCapableDevices\_ | Массив, содержащий информацию обо все GPU системы. |
| std::shared\_ptr<GPU> selectedGPU\_ | Указатель на текущий активный GPU. |

**3.3 Функциональное проектирование блока инициализации**

Данный блок будет состоять из 1 класса, который будет реализовывать шаблон проектирования «фасад». Поля и методы данного класса приведены в таблице 3.11.

Таблица 3.11 – Поля и методы класса ServiceSDK

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| 1 | 2 |
| ServiceSDK(int argc, char\*\* argv) | Конструктор. |
| ~ServiceSDK() | Деструктор. |
| void Initialize() | Инициализирует все блоки программы. |
| void Start() | Переводит потоки управляющих единиц в режим исполнения заданных алгоритмов. |
| void InitializeConfigFolderPath() | Определяет путь к папке с конфигурацией сервиса. |

*Продолжение таблицы 3.11*

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| 1 | 2 |
| void InitializeProcessingQueues(const std::shared\_ptr<Config::JsonConfig>& serviceConfig) | Инициализирует очереди для хранения результатов работы исполняющих единиц. |
| void InitializeProcessors(const std::shared\_ptr<Config::JsonConfig>& serviceConfig) | Инициализирует исполняющие единицы программного модуля. |
| std::shared\_ptr<Config::JsonConfig> GetServiceConfig() | Обращается к блоку конфигурирования, получает указатель на конфигурацию сервиса, в которой указано, сколько необходимо создать исполняющих единиц и очередей, как соединить созданные компоненты между собой. |
| static void ValidateServiceConfiguration(const std::shared\_ptr<Config::JsonConfig>& serviceConfig) | Валидирует конфигурацию сервиса на предмет соответствия формату. |
| constexpr const inline static char\* organizationName | Константа, хранящая название организации-разработчика. |
| constexpr const inline static char\* productName | Константа, хранящая название программного продукта. |
| constexpr const inline static char\* version | Константа, хранящая версию программного продукта. |
| std::unique\_ptr<Config::JsonConfigManager> configManager\_ | Указатель на управляющий класс в блоке конфигурирования. |
| std::unique\_ptr<GPU::GpuManager> gpuManager\_ | Указатель на управляющий класс в блоке управления графическими процессорами. |
| std::unique\_ptr<DataStructures::ProcessingQueueManager> queueManager\_ | Указатель на класс, управляющий очередями. |
| std::unique\_ptr<Processing::ProcessorManager> processorManager\_ | Указатель на класс, управляющий исполняющими единицами. |

*Продолжение таблицы 3.11*

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| 1 | 2 |
| void InitializeConfigManager() | Инициализирует блок конфигурации. |
| void InitializeGpuManager() | Инициализирует блок управления GPU. |
| void InitializeServiceGPU(const std::shared\_ptr<Config::JsonConfig>& serviceConfig) | Производит обращение к блоку управления GPU, получает активный графический процессор. |
| std::string configPath\_ | Путь к папке, содержащей конфигурационные файлы всех компонентов пограммы. |

Стоит отметить, что данный класс является одним из основных классов проекта. Данный класс содержит в себе указатели на объекты управляющих классов большинства блоков программного модуля, что позволяет упростить управление всей системой в целом, используя для этого средства одного класса.

**3.4 Функциональное проектирование блока управления исполняющими единицами**

Данный блок содержит набор классов, отвечающих за выполнение алгоритмов на CPU и GPU. Классы данного блока перечислены в таблице 3.12.

Таблица 3.12 – Классы блока управления исполняющими единицами

|  |  |
| --- | --- |
| Название клсса | Назначение |
| IProcessor | Базовый класс исполняющей единицы. |
| CpuProcessor | Исполняющая единица, задействующая только CPU. |
| GpuProcessor | Исполняющая единица, задействующая CPU и GPU. |
| IThread | Базовый класс исполняющего потока. |
| Thread | Класс потока, обеспечивающего однократное выполнение задачи. |
| EndlessThread | Класс потока, обеспечивающего бесконечное выполнение задачи. |
| ProcessorManager | Класс, управляющий исполняющими единицами. |

В таблицах 3.13 – 3.15 приведены описания полей и методов перечисленных классов.

Таблица 3.13 – Поля и методы класса IProcessor

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| 1 | 2 |
| IProcessor(const std::shared\_ptr<Config::JsonConfig>& config, const std::unique\_ptr<DataStructures::ProcessingQueueManager>& queueManager) | Конструктор. |
| virtual ~IProcessor() = default | Виртуальный деструктор. |
| virtual void Process() = 0 | Виртуальный метод для запуска выполнение команд. |
| virtual void Stop() = 0 | Виртуальный метод для остановки выполнения команд. |
| virtual void InitializeAlgorithms(const std::unique\_ptr< IAlgorithmFactory>& algorithmFactory, const std::unique\_ptr< JsonConfigManager>& configManager, const std::unique\_ptr<GPU::GpuManager>& gpuManager) = 0 | Виртуальный метод, инициализирующий список исполняемых алгоритмов. |
| virtual void Initialize() = 0 | Виртуальный метод инициализации. |
| virtual bool IsStarted() = 0 | Виртуальный метод, проверяющий состояние объекта. |
| const std::string& GetName() | Возвращает имя исполняющей единицы. |
| void SetName(const std::string& name) | Задаёт имя исполняющей единицы. |
| std::string name\_ | Имя исполняющей единицы. |
| std::shared\_ptr< ProcessingQueue<std::shared\_ptr< ProcessingData>>> inputQueue\_ | Указатель на очередь, из которой будут браться данные для обработки. |
| std::vector<std::unique\_ptr<Algorithms::IAlgorithm>> processAlgorithms\_ | Список выполняемых алгоритмов. |

*Продолжение таблицы 3.13*

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| 1 | 2 |
| std::shared\_ptr<DataStructures::ProcessingQueue<std::shared\_ptr<DataStructures::ProcessingData>>> outputQueue\_ | Указатель на очередь, в которую будут помещаться данные после обработки. |

Таблица 3.14 – Поля и методы класса CpuProcessor

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| 1 | 2 |
| CpuProcessor(const std::shared\_ptr<Config::JsonConfig>& config, const std::unique\_ptr<DataStructures::ProcessingQueueManager>& queueManager) | Конструктор. |
| ~CpuProcessor() override | Деструктор. |
| void Process() override | Метод для запуска выполнение команд. |
| void Stop() override | Метод для остановки выполнения команд. |
| bool IsStarted() override | Метод, проверяющий, происходит ли выполнение алгоритмов в данный момент. |
| void InitializeAlgorithms(const std::unique\_ptr<Algorithms::IAlgorithmFactory>& algorithmFactory, const std::unique\_ptr<Config::JsonConfigManager>& configManager, const std::unique\_ptr<GPU::GpuManager>& gpuManager) override | Метод, инициализирующий список исполняемых алгоритмов. |

*Продолжение таблицы 3.14*

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| 1 | 2 |
| std::shared\_ptr<DataStructures::ProcessingQueue<std::shared\_ptr<DataStructures::ProcessingData>>> outputQueue\_ | Указатель на очередь, в которую будут помещаться данные после обработки. |
| std::vector<std::unique\_ptr<Algorithms::IAlgorithm>> processingAlgorithms\_ | Список выполняемых алгоритмов. |
| void Initialize() override | Метод инициализации. |
| static void ValidateAlgorithmConfig(const std::shared\_ptr<Config::JsonConfig>& algorithmConfig) | Метод, проверяющий конфигурацию объекта на соответствие формату. |
| EndlessThread thread\_ | Исполняющий поток. |

Таблица 3.15 – Поля и методы класса GpuProcessor

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| 1 | 2 |
| CpuProcessor(const std::shared\_ptr<Config::JsonConfig>& config, const std::unique\_ptr<DataStructures::ProcessingQueueManager>& queueManager) | Конструктор. |
| ~CpuProcessor() override | Деструктор. |
| void Process() override | Метод для запуска выполнение команд. |
| void Stop() override | Метод для остановки выполнения команд. |
| bool IsStarted() override | Метод, проверяющий, происходит ли выполнение алгоритмов в данный момент. |
| cudaStream\_t cudaStream\_ | Очередь команд для GPU. |
| EndlessThread thread\_ | Исполняющий поток. |
| void Initialize() override | Метод инициализации. |

*Продолжение таблицы 3.15*

|  |  |
| --- | --- |
| Сигнатура поля/метода | Назначение |
| 1 | 2 |
| void InitializeAlgorithms(const std::unique\_ptr<Algorithms::IAlgorithmFactory>& algorithmFactory, const std::unique\_ptr<Config::JsonConfigManager>& configManager, const std::unique\_ptr<GPU::GpuManager>& gpuManager) override | Метод, инициализирующий список исполняемых алгоритмов. |
| static void ValidateAlgorithmConfig(const std::shared\_ptr<Config::JsonConfig>& algorithmConfig) | Метод, проверяющий конфигурацию объекта на соответствие формату. |